情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 2　76–87 (June 2010)

通信アルゴリズム評価用 メッセージフローシミュレータの開発

矢崎俊志[†1]　石畑宏明[†2]

並列計算機の通信アルゴリズムの性能評価を行うためのシミュレータ MFS（Message Flow Simulator）を開発した．MFS は，既存のパケットレベルシミュレータと異なり，並列計算機の相互結合網を構成する通信路のバンド幅と，そこを流れるデータの流量に基づくシミュレーションを行う．本論文では 3,000 ノード規模の Fattree ネットワーク上での全対全通信において，MFS の特性を評価した．100 ノードから 256 ノードの 2 次元 Torus ネットワークにおいて，イリノイ大学で開発された BigSimulator とシミュレーション結果を比較した．MFS の見積もった通信時間は，BigSimulator に対して平均で約 40%小さかった．この比較実験により，MFS のシミュレーション精度は，BigSimulator より低いが，現実的でない見積りを与える理論的最小値より実機に近いことを示した．並列化していない MFS のシミュレーション実行時間は，7 CPU コアで並列実行した BigSimulator の約 1/127 から約 1/337，平均で約 1/188 であった．16×16 の 2 次元 Torus 上で，3 種類の全対全通信アルゴリズムについて，MFS と BigSimulator の両方で，その性能差を同様に評価できることを示した．

Development of a Message Flow Simulator for Evaluation of Communication Algorithms

Syunji Yazaki[†1] and Hiroaki Ishihata[†2]

This paper proposes Message Flow Simulator (MFS), which evaluates the communication algorithms for interconnection network of large-scale parallel computer. MFS calculates communication time from the amount of message flow on communication links. We evaluate characteristics of MFS on Fat-tree network including up to 3000 nodes for All-to-all communication. We also compared MFS with BigSimulator that employed packet-based simulation and was developed at University of Illinois. On virtual 2D-Torus network including 100 to 256 nodes for All-to-all communication, non-parallelized MFS reduces simulation run time by 1/127 to 1/337 (1/188 in average) from run time of parallelized BigSimulator that is executed by 7 CPU-cores. MFS gives more accurate results than the theoretical minimum value of communication time for All-to-all communication and less accurate results than BigSimulator. Difference of estimated communication time between MFS and BigSimulator was 40% in average. MFS evaluates performance of communication algorithms on 16×16 2D-Torus network same as BigSimulator.

1. はじめに

1.1 背　景

近年，ノード間通信を必要とする並列計算において，通信時間の増加に起因する性能向上率の低下が問題となっている．ノード間通信が必要な並列計算で多ノードを用いる場合，通信量や回数が少ノードの場合より多くなるため，通信時間が並列プログラム実行のボトルネックとなる可能性がある．近年の並列計算機は数万ノードを相互接続網で接続したものである．このノード数は，将来的に数十万に達する見込みであり，通信時間に関する問題は今後ますます大きくなることが予想される．

相互接続網の実装コストは，ノード数の増加にともなって増える．この増加度合いは相互接続網の構造によって大きく異なる．初期の地球シミュレータ[1] など，比較的ノード数が少ない並列計算機は，通信経路上で競合が発生しないクロスバなどで相互接続網を構成することができた．しかし，数万から数十万ノードをこれで接続することは構造やコストの面から現実的ではない．近年では数千ノードの接続に，よりコストの低い Fattree トポロジの相互結合網を採用している例がある．数十万ノード規模の並列計算機では，さらに，Mesh，Torus，バンド幅の小さい Fattree，またはこれらを組み合わせた不均一なトポロジを選択せざるをえない．このようなトポロジで構成された相互結合網は，バイセクションバンド幅が狭く，通信経路の競合も起きやすい．この競合が通信効率低下の原因となる．不均一なトポロジの相互結合網を用いている例として，Red Storm のように，z 次元のみを Torus 状にした 3 次元 Mesh を採用し，外周のノードに I/O を備えたものがある[2]．また，Cray TX4 のように Mesh と Torus を組み合わせたもの[3]，Roadrunner のように，ハーフバイセクションバンド幅の Fatree を用いたもの[4] などもある．

通信路の競合を可能な限り回避し，狭いバンド幅を有効活用するために，様々な通信アル

†1 電気通信大学
The University of Electro-Communications
†2 東京工科大学
Tokyo University of Technology

ゴリズムが考案されている．従来，通信アルゴリズムは，通信競合が少ないことを前提に，より広い適用範囲を求め，トポロジに依存しないものとして開発されてきた．またその評価は，通信アルゴリズムを適用した場合の通信時間の理論値を通信の理論的最小値と比較することにより行われてきた．この通信の理論値や理論的最小値は，解析的な手法によって求められてきた．

通信競合が起きやすいトポロジを持つ近年の並列計算機向けには，特定のトポロジや通信パターンに合わせた最適な通信アルゴリズムが考案されている．これらの通信アルゴリズムの多くは複雑であり，評価に用いる通信時間の理論値を解析的に求めることが難しい．

複雑な通信アルゴリズムの評価はシミュレーションにより行われてきた．このシミュレーションは，通信経路の競合を考慮したうえで，全通信の完了時間を見積もるものである．このようなシミュレーションには，パケットやさらに細かいフリット単位での通信シミュレーションを行うパケットレベルシミュレータが主に用いられてきた．

パケットレベルシミュレータによる大規模ネットワークのシミュレーションには，通信されるパケット数に比例した時間がかかる．よって，より大規模なネットワークのシミュレーションに対応した高い拡張性と，様々な通信パターンのシミュレーションを短時間で実行できる高速性を備えたシミュレータが求められている．

1.2 目　　的

本研究では，大規模並列計算機向けの通信アルゴリズムを評価することを目的とし，高い拡張性と高速性を実現したシミュレータ Message Flow Simulator（MFS）を提案，実装する．MFS は，通信を連続体の流れとして扱い，その流量が通信経路の競合により制限されるというモデルに基づくシミュレーションを行う．この方式は，パケットの振舞いを詳細に追跡する既存の方式より高速に通信時間を見積もることができる．これにより，大規模ネットワークのシミュレーションを短時間で行うことができる．

本論文では，2 章で関連研究を紹介する．3 章で MFS のシミュレーション方式とその実装を述べる．4 章では数値実験により，MFS の特性を評価する．5 章では MFS のシミュレーション結果と実行速度を既存のパケットレベルシミュレータと比較する．6 章では MFS とパケットレベルシミュレータによる通信アルゴリズムの評価結果を比較する．最後に 7 章でまとめを述べる．

2. 関 連 研 究

過去に並列計算機の相互接続網上で発生する輻輳とその影響を調べるために，様々な通信シミュレータが開発されている．これらの多くは，パケットやフリットなどの単位で精密なシミュレーションを行うパケットレベルシミュレータである．並列計算機を対象としたシミュレータは，並列計算機の相互接続網の特徴である，(1) 規則正しいトポロジ，(2) バンド幅はあるまとまりにおいて同じであり，通信遅延が小さい，(3) 通信は規則的に発生する，(4) 主にロスレス網である，という性質を考慮して設計されている．これに関連して，汎用の離散イベントシミュレーション環境を用いたシミュレーション[5] や，シミュレーション環境そのものの開発が行われている[6),7)]．シミュレータの実装を支援するため，相互接続網の構造をプログラムによって構築することができる環境を開発している例もある[8),9)]．

これらの研究では，シミュレーションによる評価対象がスイッチなどのネットワーク機器や，各ノードでの計算時間を含んだ並列プログラムの全実行時間である．そのため，相互接続網を構成するスイッチなどの装置を詳細にモデル化している．これにより，ハードウェアの動作に近いシミュレーション結果が期待できる一方で，大規模のシミュレーションには多くの時間を必要とする．また，イベントシミュレーションでは，発生したイベントの同期をとりながらシミュレーションが進むため，並列化による実行時間の短縮が難しい．これに対して分散メモリマシン上で投機的にイベント処理を実行することで，高い並列度を実現した BigSimulator[10)–13)] も開発されている．

並列計算機の相互接続網を対象としたシミュレータは，通信性能の評価だけでなく並列プログラムの性能予測にも用いられている．並列プログラムの実行時間は，計算時間と通信時間に分けて予測される場合が多い．これまで，実測値に基づいて通信時間の予測を行った例がある[14)]．近年では，これにパケットレベルシミュレータが多く用られている[15),16)]．

MFS をこのような並列プログラムの性能予測に用いることもできる．シミュレーション実行時間が現実的な範囲であれば，より実機に近い振舞いをするパケットレベルシミュレータを並列プログラムの性能予測に用いることは有効である．しかしそうでない場合には，MFS を性能予測に用いるという選択肢もある．この場合，高速な MFS を用いることで，今まで評価できなかった規模の並列プログラムの性能を評価することが可能となる．

性能予測に関する研究では，単に性能の予測を行うだけでなく，並列プログラムの改善点を指摘するという試みも行われている．神戸らは，並列プログラムのボトルネックを指摘するツールの開発を行っている[17)]．このツールでは，通信遅延，各ノードでの通信オーバヘッド，メッセージの送受信間隔，プロセッサ数，通信の同期コストを考慮したモデルで並列プログラムの性能予測行い，その結果に基づいて，並列プログラムのボトルネックを判定している．並列計算機の相互結合網自体の性能を評価する研究に関しては，多段結合網の転送性

能を解析的に求めている例や[18]，メッセージ転送の信号遅延を評価するためのネットワークシミュレータを Java で開発している例がある[19]．

インターネットを構成する TCP/IP 網を対象としたシミュレータには，本論文で提案するような，通信の流量に基づくシミュレータがいくつかある．Nicol は，IP 網上の TCP パケットを流量としてモデル化している[20]．最近では，固定時間ステップで流量シミュレーションを行う方法も提案されている[21]．また，TCP/XCP/UDP の各プロトコルについて流量シミュレーションを行うシミュレータも開発されている[22]．インターネットを構成する TCP/IP 網の特徴は，並列計算機の相互接続網とは異なり，(1) ルータを階層的に接続した不均一なトポロジ，(2) バンド幅は区間によって異なり，遅延時間も大きい，(3) 通信はランダムに発生する，(4) 主にロス網である，という性質がある．このようなネットワークでは，スケジューリングやフロー制御の方法，またはプロトコルの性能に基づく 1 対 1 の通信性能が重要となる．よって，これらのシミュレータは，1 対 1 の通信性能の評価に重きをおいて開発されている．これを並列計算機の相互接続網のシミュレーションに用いるためには，大きな変更が必要になる．

3. Message Flow Simulator（MFS）

3.1 シミュレーション手法

通信アルゴリズムの性能は，通信競合の度合いで評価することができる．通信アルゴリズムの設計においては，通信時間を短縮するために，通信のタイミング，順番，経路を変えることで，通信競合の度合いを下げる工夫を行う．この競合度合いの評価に既存のパケットレベルシミュレータを用いることも可能である．しかし，パケットを個々に追跡するこの手法は，パケット数の増加にともなってシミュレーション実行時間が長くなるため，大規模な並列計算機（並列プログラム）の評価には多くの時間を必要とする．

これに対して MFS は，通信を流れ（フロー）として扱い，通信競合の度合いを評価する．現実の通信において，メッセージ通信がある経路で競合すると，その経路では，各メッセージを構成するパケットが混ざりあって流れる．MFS は，このような競合をフローの重なりとしてとらえる．重なりによって，その経路を共有して流れる全フローの流量が制限される．制限された流量とメッセージサイズから，通信が完了するまでの時間を求めることができる．

この手法は，パケットレベルシミュレーションと比較して，シミュレーション実行時間を短縮することができる．パケットレベルシミュレーションでは，個々のパケットの動きを追跡するため，実行時間がパケット数に依存する．一方，MFS ではメッセージごとにその流れをシミュレーションするため，実行時間はメッセージ数に依存する．多くのメッセージは，複数のパケットで構成されるため，一般にメッセージ数はパケット数よりも少ない．よって，MFS の計算時間はパケットレベルシミュレータの計算時間よりも短くなる．

MFS のシミュレーション方式では，既存のパケットレベルシミュレータのように，バッファリングやアービトレーション機構など，ハードウェアの詳細な振舞いを実機に忠実にモデル化しない．これにより，通信アルゴリズムの性能に主として影響する経路競合の度合いのみを高速に評価する．一方で，実機の動作に近いという意味での精度はパケットレベルシミュレータより低くなる．

3.2 ネットワーク，ノードおよび通信のモデリング

MFS は，相互接続網を複数のノード，スイッチおよびこれらを接続するリンク（通信路）でモデル化している．各ノードは，通信されるデータをメッセージとして持つ．全メッセージは，自身の宛先とサイズを情報として持つ．各ノードにある送信待ちの全メッセージは各ノードがそれぞれ持つキューに保存されている．各ノードは，送信待ちメッセージとは別に，送信中または次のタイミングでただちに送信可能なメッセージを持つ．このメッセージをアクティブメッセージと呼ぶ．各スイッチは異なる数の入出力ポートを持つことができる．各スイッチはバッファを持たない理想的なクロスバスイッチとする．各スイッチでのルーティングは静的に行う．リンクは向きを持つ．双方向通信には，逆向きのリンクを 2 つ用いる．

通信は次のようにモデル化されている．アクティブメッセージから生成される通信の流れをフローと呼ぶ．フローは，リンクやスイッチを通って，宛先へ向かって流れる．複数のフローが，同じリンクや同一スイッチ内の同じ出力ポートを通る場合，そのリンクやスイッチの出力ポートにあらかじめ設定されたバンド幅は，そこを通るすべてのフローに対して公平に割り当てられる．割り当てられたバンド幅は，各フローが利用可能なバンド幅であり，これがフローの流量となる．

通信における送受信のモデルは次のとおりである．送受信において，送信側は一時に送信されるメッセージの数を限定するためにブロッキング送信を行う．受信側では，すべてのフローを同時に受信するためにノンブロッキング受信を行う．ただし，MFS はフローの重なりによって通信の流量を制限するというモデルを用いているため，同時に受信しているメッセージの数だけ，個々のメッセージを受信するためのバンド幅は減少する．

現実の通信では，同期された通信と非同期の通信の両方が発生する．MFS において，非同期の送受信は次のようにモデル化されている．送信ノードは，自身が送信する全メッセー

ジをメッセージキューに持ち，一時に 1 個のメッセージをキューから取り出す．取り出されたメッセージは，アクティブメッセージとなり，送信される．送信が完了するとキューから次のメッセージが取り出され，新たなアクティブメッセージとなる．受信ノードは，送信ノードからメッセージが送出された瞬間から受信を開始する．受信ノードは，そのバンド幅の範囲内で，複数のメッセージを同時に受信することができる．送信と受信の同期は，次の処理により実現されている．送信ノードのキューと受信ノードを対応付ける．送信ノードは，自身のキュー上に受信待ちメッセージを見たら，そのメッセージが受信されるまで次の送信を行わない．

3.3 MFS の処理概要と実装

MFS は，フローの重なりによってその流量が制限されるモデルに基づいて，通信に必要な時間を計算する．計算の手順を次に示す．

(1) 与えられたトポロジと通信パターンで，全ノードペア間のフローと，その経路のリストを作る．
(2) 相互接続網中の全リンクについて，各リンクを通過するフローの重なりの数を数える．
(3) フローの重なりの数から，各リンクを通るフロー 1 つあたりが利用できるバンド幅を算出する．算出において，バンド幅はリンクを共有する全フローで等しく分けられる．重なるフローの数が多ければ，その分だけ 1 つのフローが利用できるバンド幅は小さくなる．
(4) 算出した各フローのバンド幅と各メッセージサイズから，各メッセージの通信に必要な時間を求め，その時間分だけシミュレーションの時間を進める．
(5) 上記の処理を，与えられた通信パターンに含まれる全通信が完了するまで繰り返す．

手順の概要を**図 1** に示す．図 1 中の各記号について，その意味をまとめ，**表 1** に一覧を示す．ここに，η を全ノード数，ω を全メッセージ数，ι を全リンク数（η，ω，ι は非負整数）とする．並列計算機を構成するノードの集合を $N = (n_i)_{i=0}^{\eta-1}$ とおく．以降，添字 i は，i 番ノード n_i に関連する記号に対して共通に用いる．全送信待ちメッセージの集合を $WM = (wm_j)_{j=0}^{\omega-1}$ とおく．添字 j は，同様に，j 個目の送信待ちメッセージ wm_j に関連する記号に対して共通に用いる．前節に述べたモデルでは，送信待ちメッセージは，各ノードのキューに保持されいてる．WM はこれら全送信待ちメッセージを要素とする集合として定義する．各ノードは，WM に含まれる送信待ちメッセージの中で，自身が送信を担当するメッセージだけを自身の送信待ちメッセージとしてキューに保持しているものと考える．次に，各ノード n_i におけるアクティブメッセージを m_i と表現する．たとえば，0 番ノード n_0 のアクティブメッセージを m_0 のように表現する．全ノードのアクティブメッセージの集合を $M = (m_i)_{i=0}^{\eta-1}$ とする．アクティブメッセージ m_i が生成するフローの集合を $F = (f_i)_{i=0}^{\eta-1}$ とする．ここに，フロー f_i が利用可能なバンド幅を $fb_i \in FB$ とする．また，m_i について，そのサイズを $ms_i \in MS$，送信を終えるために必要な時間を $mt_i \in MT$ とそれぞれおく．

ノードもしくはスイッチをつなぐリンクの集合を $L = (l_k)_{k=0}^{\iota-1}$ とおく．以降，添字 k は，添字 j と同様の考え方で，k 番リンクに関連する記号に対して共通に用いる．l_k に対して，設定された物理バンド幅を $pb_k \in PB$ とする．たとえば，0 番リンク l_0 の物理バンド幅を pb_0 のように表記する．各リンクを通過するフローが利用できる実効的なバンド幅を

Init　送信すべきメッセージの集合 WM を用意する．ユーザに指定されたトポロジ中の $\forall l_k \in L$ について，物理バンド幅 pb_k を設定する．プログラムの収束条件 th を設定する．
Step 1　$WM = \phi$ ならば終了する．$\forall n_i \in N$ の m_i について，$ms_i \leq 0$ ならば，次に送信するメッセージを WM から選び，新たな m_i とする．新たな m_i に対応する ms_i を新たなメッセージサイズとする．
Step 2　$\forall l_k \in L$ について，$c_k = 0$ とする．
Step 3　$\forall n_i \in N$ について，m_i から f_i を求める．さらに，$\forall f_i \in F$ について，$\forall l_k \in LinkSetIn(f_i)$ に対応するカウンタ c_k の値を 1 増やす．
Step 4　$\forall l_k \in L$ について，$eb_k = pb_k / c_k$ とする．
Step 5　$\forall l_k \in L$ について，$pb_k - \{\sum fb_i \in FlowSetOn(l_k)\}$ の変化が th 以下になるまで，**Step 5.1** から **Step 5.4** をループ処理として，繰り返し実行する．

ループ処理
Step 5.1　$\forall f_i \in F$ について，$\forall l_k \in FlowSetOn(f_i)$ に対応する eb_k の中から最小のものを求め，fb_i とする．
Step 5.2　$\forall f_i \in F$ について，$\forall fb_i$ を $l_k \in LinkSetIn(f_i)$ に対応する eb_k に代入する．
Step 5.3　$\forall l_k \in L$ について，$\forall f_i \in FlowSetOn(l_k)$ に対応する fb_i の総和 $\sum fb_i$ を求める．
Step 5.4　$\forall l_k \in L$ について，$(pb_k - \sum fb_i) \neq 0$ ならば，$\forall fb_i \in FlowSetOn(l_k)$ について，$fb_i = fb_i + (pb_k - \sum fb_i)/c_k$ とする．

Step 6　$\forall m_i \in M$ について，$mt_i = ms_i / fb_i$ とする $(mt_i \in MT)$．
Step 7　$\min MT$ を求め，$\min MT < Resolu$ ならば $\Delta t = Resolu$ とし，そうでなければ $\Delta t = \min MT$ とする．
Step 8　$T = T + \Delta t$ を行う．また，$\forall m_i \in M$ について，各メッセージが Δt 時間分だけ送られたものとし，$ms_i = ms_i - (fb_i \times \Delta t)$ とした後，**Step 1** へ戻る．

図 1　MFS によるシミュレーションの概要
Fig. 1　Outline of processing for simulation by MFS.

表 1　図 1 中の記号一覧
Table 1　A list of signs in Fig. 1.

η	全ノード数	Δt	1 サイクルの時間
ω	全メッセージ数	$Resolu$	時間分解能（Δt の最小値）
ι	全リンク数	T	サイクル時間の合計（経過した時間の累計）
		th	Step 5 のループ処理の収束条件となるしきい値
n_i	i 番ノード ($i = 0..\eta - 1$)	N	ノード n_i の集合
m_i	i 番ノードのアクティブメッセージ ($i = 0..\eta - 1$)	M	アクティブメッセージ m_i の集合
ms_i	メッセージ m_i のサイズ ($i = 0..\eta - 1$)	MS	メッセージサイズ ms_i の集合
mt_i	サイズ ms_i のメッセージ送信に必要な時間 ($i = 0..\eta - 1$)	MT	要求メッセージ送信時間 mt_i の集合
f_i	メッセージ m_i が生成するフロー ($i = 0..\eta - 1$)	F	フロー f_i の集合
fb_i	f_i が利用可能なバンド幅 ($i = 0..\eta - 1$)	FB	フローが利用可能なバンド幅 fb_i の集合
wm_j	j 個目の送信待ちメッセージ ($j = 0..\omega - 1$)	WM	全ノード中の送信待ちメッセージ wm_j の集合
l_k	k 番リンク ($k = 0..\iota - 1$)	L	リンク l_k の集合
pb_k	l_k に設定された物理バンド幅 ($k = 0..\iota - 1$	PB	リンクの物理バンド幅 pb_k の集合
eb_k	l_k の実効バンド幅 ($k = 0..\iota - 1$)	EB	リンクの実効バンド幅 eb_k の集合
c_k	l_k を通過するフローの数を数えるカウンタ ($k = 0..\iota - 1$)	C	通過フロー数のカウンタ c_k の集合
添字 i, j, k はノードやリンクと，その要素の対応関係を示す.		$LinkSetIn(f_i)$	フロー f_i が通過するリンク l_k の集合
0 番ノード n_0 のアクティブメッセージを m_0 のように表現する.		$FlowSetOn(l_k)$	リンク l_k を通過するフロー f_i の集合

$eb_k \in EB$ とおく．l_k を通過するフローの数を数えるカウンタの集合を $C = (c_k)_{k=0}^{\iota-1}$ とする．$LinkSetIn(f_i)$ はフロー f_i が通過するリンク l_k の集合を表す．$FlowSetOn(l_k)$ はリンク l_k を通過するフロー f_i の集合を表す.

処理は，図 1 の Step 1 から Step 8 を 1 サイクルとして，すべてのメッセージの送信が終了するまで繰り返される．ただし，Step 5 には，ループ処理が含まれる．1 サイクルの時間を Δt で表す．Δt はフローの重なりによって変わり，重なりが多い場合には，小さな値となる．このとき，シミュレーションは非常に短い時間を 1 サイクルとして進行するため，結果として処理時間が長くなる．Δt が小さくなりすぎることを防ぐために，Δt の最小値を時間分解能 $Resolu$ として定義する．サイクル時間の累計が経過したシミュレーション時間となる．これを T で表す．さらに，Step 5 における収束条件として，しきい値 th を設ける．

Init では，各値の初期化やメッセージの準備，収束条件の設定などを行う．Step 1 では，まず，各ノードで送信中のアクティブメッセージの残りサイズを調べる．アクティブメッセージの残りサイズが 0 であるノードは，キューから次にアクティブにするメッセージを取り出す．すべてのメッセージの送信が終了していれば，シミュレーションを終了する．

Step 2，3，4 では，メッセージが生成するフローの一覧を作っている．さらに各リンクを通るフローに割り当てるバンド幅を計算するため，あるリンクを同時に通るフローの数を数え，各リンクに設定された物理バンド幅をその数で割ったものをリンクの実効バンド幅としている．各フローが利用できるバンド幅は，そのフローが通過する全リンクの実効バンド幅の最小値となる．

Step 5 では，このように算出したフローのバンド幅を初期値として，ループ処理により，物理バンド幅を使いきっていないリンクを探している．各リンクで使いきれずに余ったバンド幅は余裕のある他のフローに割り当てられる．この処理を，全リンクの物理バンド幅が 1 である場合を例に説明する．今，リンク l_0 を共有するフロー f_0 と f_1 が存在する．f_0 のバンド幅が，別のリンク l_1 ですでに 1/3 に制限されており，f_1 のバンド幅は他のどのリンクでも制限されていない場合を考える．l_0 では，f_0 と f_1 の両方に 1/2 のバンド幅を割り当てることができる．しかし，f_1 はすでに l_1 で 1/3 に制限されているため，l_0 から割り当てられたバンド幅 1/2 を活用することができない．このような場合に，物理バンド幅を無駄なく使うため，f_1 が使いきれない分のバンド幅を f_0 に割り当てる．今の場合，f_1 が l_0 使いきれなかったバンド幅 1/6（$= 1/2 - 1/3$）を f_0 に割り当てることができる．

Step 6，7 では，フローのバンド幅から，各ノードペア間を流れるメッセージの転送に必要な通信時間を求めている．全ノードペアのアクティブメッセージの送信に必要な時間を求

め，その最小値を次に進める 1 サイクルの時間 Δt としている．なお，サイクルの時間が不必要に細かくならないようにするため，その最小値として時間分解能 $Resolu$ を設定しておく．最悪の場合では，この時間分解能の単位でシミュレーションが進む．時間分解能を，小さくすればシミュレーションの精度は良くなるが，その分，実行時間が長くなる．Step 8 では，求めた 1 サイクルの時間分だけ各ノードのアクティブメッセージが送信されたものとし，各メッセージサイズを減じている．

3.4 MFS の主な機能

MFS の主な機能を述べる．MFS は，シミュレーションの対象とする相互接続網と通信に関して，トポロジ，通信パターン，通信アルゴリズム，メッセージサイズ，メッセージ数などを指定することができる．トポロジとして，ハイパークロスバ，Fattree，Torus，Mesh がある．各トポロジにおいて，スイッチの段数，次元数，および次元ごとのバンド幅を指定することができる．典型的な通信パターンとして，全対全，Ring，Broadcast などを扱うことができる．実装された通信アルゴリズムの一例として Simple-Spread 法や Pairwise 法がある．メッセージサイズとメッセージ数を指定することで，大きなメッセージが少し流れた場合や，小さなメッセージが多く流れた場合など，様々な通信パターンについてシミュレーションを行うことができる．

MFS は，PSI-SIM[16] で行われているような，実際のアプリケーションの通信ログに基づくシミュレーションも行うことができる．これには，パターンファイルを用いる．パターンファイルでは，メッセージの通信に関して，メッセージを送信するノード番号，宛先，送信の順番，サイズなど指定することができる．パターンファイルを，アプリケーションの通信ログに基づいて構成することで，実際のアプリケーションが生成する通信パターンのシミュレーションを行うことができる．パターンファイルでは，通信の待ち時間も指定することができる．これによってメッセージの送信を指定時間だけ遅らせることができる．この機能により，並列プログラムの各ノードにおける計算時間や，通信の立ち上がり時間を考慮したシミュレーションを行うこともできる．

4. MFS の特性評価

4.1 実 験 方 法

MFS によるシミュレーションの特性を示すため，シミュレーションによって見積もられる通信時間と，シミュレーション実行時間を調べる数値実験を行った．実験は，$2p$ 個のポートを持つスイッチ 3 段で構成されたフルバイセクションバンド幅の Fattree ネットワーク（以降，FBB Fattree）について行った．このネットワークの構成は次のとおりである．まず，$2p$ 個の下位向きポートを持つスイッチを最上の 3 段目に配置する．続く 2，1 段目にはそれぞれ上位向きに p ポート，下位向きに p ポートを持つスイッチを配置する．ノードは，1 段目の下向きポートに接続されている．

通信パターンとして，全対全通信を用いる．全対全通信では，各ノードが自分以外の全ノードに対してメッセージを送受信するため，全ノード数を $Nnode$ とした場合，1 ノードあたり $Nnode - 1$ 回の送信を行う．これは，ネットワーク負荷の高い通信パターンであり，FFT や行列の転置処理などでよく現れる．全対全通信の通信アルゴリズムとして，ノード数が 2 のべきの場合には Pairwise 法を用いた．それ以外の場合には，Simple-Spread 法を用いた．この方法は，MPI ライブラリの MPICH[23] で，大きなサイズのメッセージを通信する場合に用いられる方法である．

シミュレーションにあたり，全リンクの物理バンド幅と全メッセージのサイズを 1 に設定した．MFS において，物理バンド幅とメッセージサイズの単位は任意である．MFS は，通信時間を，物理バンド幅とメッセージサイズから求めている．よって，MFS の見積もる通信時間の単位は，これらの単位で決まる．以降，本文では，シミュレーション上の通信時間と実時間を区別するため，前者を仮想通信時間（Virtual communication time）と呼ぶ．

仮想通信時間の測定は，全部で 4 通り行った．並列プログラム実行時に各ノードへの Rank 割当てがランダムである場合とそうでない場合を考え，ノード番号の配置を規則正しくした場合とそうでない場合を想定する．このそれぞれに対して，図 1 の Step 5 に示すループ処理の効果を調べるため，ループ処理を行った場合とそうでない場合について，仮想通信時間を測定した．同時に，シミュレーション実行時間も測定した．測定には，PRIMERGY RX200S3 サーバ（Intel Xeon 3.0 GHz Dual-core 2-processor，8 GByte memory）を用いた．

4.2 ノード数と仮想通信時間

MFS が見積もった仮想通信時間を**図 2** にまとめる．図中のグラフは，ノード数（$Nnode$）と仮想通信時間の関係を示している．Mode=N は，図 1 の Step 5 のループ処理において，収束条件 th を 1%以下とした場合の測定結果であることを示す．Mode=F は，ループ処理を行わない場合の測定結果であることを示す．Regular と Random は，Rank 割当てが規則正しく行われている場合とそうでない場合を表す．

Rank 割当てが規則正しい場合（Regular）に着目すると，Mode=N（×），Mode=F（+）ともに，全対全通信における仮想通信時間がノード数の 2 乗に一致していることが分かる．

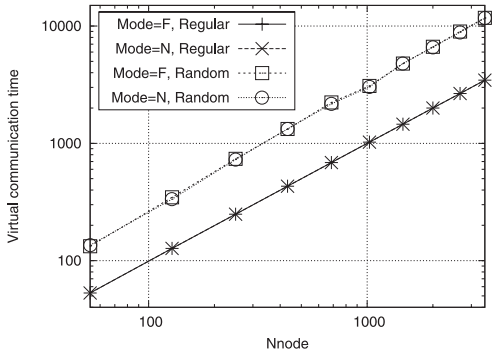


図 2　全対全通信におけるノード数と仮想通信時間の変化
Fig. 2　The number of node vs. virtual communication time for All-to-all.

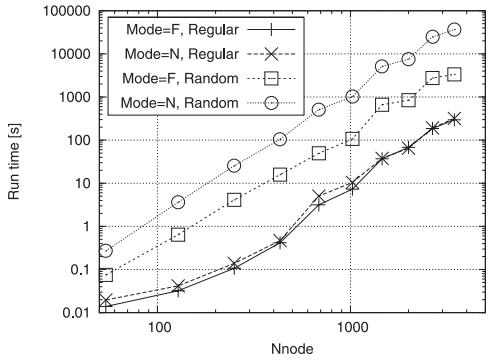


図 3　全対全通信におけるノード数とシミュレーション実行時間の変化
Fig. 3　The number of node vs. All-to-all simulation run time.

これは，理想的な全対全通信時間である．全対全通信は，n 個のノードによって，全部で n^2 回（自身へのメッセージ送信を数えない場合は $n(n-1)$ 回）の通信が発生する．FBB Fattree 上では，Rank 割当てが規則正しく行われていると，経路の競合はいっさい発生しない．そのため，測定結果が示すように，全対全通信は理想的な時間で終了する．

次に，Random の場合に着目する．この場合も，Mode=N（○）と Mode=F（□）のグラフはほぼ重なっている．この差は 0%〜4%であった．Mode=N と Mode=F の差は，Step 5 のループ処理を実行する場合と実行しない場合の差である．ループ処理を実行しないと，リンクによっては，物理バンド幅を最大限活用できない．このため，一般的には仮想通信時間は長くなる．しかし，今回の実験では，差はほとんどなかった．これは，Mode=N において，ループ処理の初期値として使用した値，すなわちリンクの物理バンド幅を，そこを流れるフローの数で公平に分割したものが良い近似値を与えているためであると考える．

Rank 割当てが規則正しい場合（× と +）とそうでない場合（○と□）では，仮想通信時間に 2.5 倍（$Nnode = 54$）から 3.4 倍（$Nnode = 3456$），平均すると約 3.1 倍の差があった．この差は，Rank 割当てを不規則にしたことによって発生した通信経路の競合が，通信効率に及ぼす影響を示したものである．

4.3　ノード数とシミュレーション実行時間

前節のシミュレーションにおける実行時間の比較を**図 3** に示す．図の見方は，仮想通信時間の比較と同様である．ただし縦軸はシミュレーション実行時間を示す．

図中のグラフが示すすべてのシミュレーション実行時間は，ノード数の 2 乗にほぼ比例している．MFS によるシミュレーションの処理量はネットワーク中の全リンク数と送信すべき全メッセージ数の両方に比例する．FBB Fattree におけるリンク数はノード数に比例する．全対全通信で送信される全メッセージ数はノード数の 2 乗である．よって，シミュレーション実行時間のオーダはノード数の 2 乗になる．このことが実験の結果からも確認できた．

また図から，Regular における Mode=N（×）と Mode=F（+）の実行時間の差はほとんどないことが分かる．前述で示した仮想通信時間の比較では，Regular の Mode=F と Mode=N にはまったく差がなかった．これは，Step 5 のループ処理が実質的に行われていないことを示している．よって，実行時間にもほとんど差は現れない．

一方，Random における Mode=N（○）と Mode=F（□）との実行時間の差は約 3.6 倍（$Nnode = 54$）から約 11 倍（$Nnode = 3456$）倍，平均で約 7.8 倍であった．このときの Step 5 におけるループ処理の反復回数は，8〜13 回程度であった．このループ処理にかかる時間が，処理時間が差となって現れている．今回行った FBB Fattree 上の全対全通信の場合においては，図 2 に示したように，Mode=F でも十分な精度が得られる．よって，平均で約 7.8 倍の実行時間を必要とする Mode=N で，より正確な値を得る価値は少ない．

5.　BigSimulator および理論的最小値との比較

5.1　実 験 方 法

ここでは，MFS と既存のパケットレベルシミュレータのシミュレーション結果およびシミュレーションの実行時間を比較する．既存のパケットレベルシミュレータとして，BigSimulator

を用いる．

この実験では，BigSimulator のシミュレーション結果が，実機に近いものであるという仮定のもとに結果の比較を行う．BigSmulator は，ハードウェアを精密にモデル化することで，実機に近いシミュレーション結果を得ることができる．特に，並列アプリケーションの実行時間の予測においては，高い予測精度を示すことが報告されている[⋆1,13]．

実験における BigSimulator と MFS の動作条件を述べる．BigSimulator での測定においては，並列計算機のテンプレートモデルとして，BlueGene モデルを用いた．これは，BigSimulator にあらかじめ用意されたものである．このモデルは，相互結合網のトポロジとして 3 次元 Torus を用いている．実験では，z 次元の大きさを 1 とし，$n \times n$ の 2 次元 Torus として用いた．通信パターンは全対全通信とし，通信アルゴリズムには，4.1 節で述べた，Pairwise 法と Simple-Spread 法を組み合わせたものを用いた．ネットワーク中の全リンクのバンド幅は 1 GByte/s とし，メッセージサイズは 20 KByte とする．最大パケットサイズは 1500 Byte とした．パケットをできるだけスムーズに配送するため，相互結合網を構成するスイッチのバッファサイズは最大パケットサイズに対して十分に大きくした．

実験ではメッセージがネットワーク上を流れている時間のみを比較する．BigSimulator は，メッセージの流れだけでなく，ノードで行われる通信の初期化や終了に関わる処理もシミュレーション結果に含める．そこで，BigSimulator による仮想通信時間の測定においては，このようなノードでの処理時間を取り除くため，次のような測定を行った．まず，実行に関わる最初のイベントの開始時刻と，全イベントの終了時刻との差を，BigSimulator が提供する Projections と呼ばれるツールで測定し，それを，通信を含めたすべての処理に要した時間とする．この測定を，メッセージサイズを 1 Byte とした場合と 20 KByte とした場合の 2 回行う．この差を BigSimulator が見積もる仮想通信時間とした．MFS は元々，ネットワーク上のデータの流れのみをシミュレーションの対象としているため，測定に特別な工夫は必要ない．MFS での測定においては，ループ処理を行わない Mode=F を用いた．送信するメッセージの量を BigSimulator に合わせるため，メッセージサイズを 20 とし，これを 20 KByte とおいた．リンクのバンド幅は 1 とし，これを 1 GBtye/s とおいた．

両シミュレータが見積もった仮想通信時間を，解析的に求めた通信時間の理論的最小値とも比較する．ノード数 n^2（$= n \times n$）の 2 次元 Torus における全対全通信時間の理論的最小値は，バイセクションバンド幅を $2n$ として，$\frac{n^3}{8}(= \frac{\frac{n^2}{2} \times \frac{n^2}{2}}{2n})$ で求まることが知られている．この式から，ノード数に応じた通信時間の理論的最小値を求めた．

MFS と BigSimulator のシミュレーション実行時間を比較するため，仮想通信時間の測定時に，各シミュレータの実行時間を測定した．BigSimulator の実行時間は，メッセージサイズを 20 KByte とした場合の実行時間を比較に用いる．測定に用いた計算機は Dell PowerEdge SC1435（AMD Quad-Core Opteron 2350 × 2，32 GByte memory）である．BigSimulator は 1 回のシミュレーションを並列化し，高速に行う機能を備えている．実験に用いた計算機には合計 8 個の物理的なプロセッサコアが実装されている．BigSimulator による実験では，1 回のシミュレーション実行時間をできるだけ短くするため，1 回のシミュレーションを 7 コアで実行した．8 コアを用いなかったのは OS の実行に必要な他のプロセスの実行を阻害しないためである．MFS は並列化を行っていないため，1 コアで実行した．ただし，実行には，高速な Mode=F を用いた．

5.2　シミュレーション結果

MFS と BigSimulator が見積もった仮想通信時間および，解析的に求めた全対全通信時間の理論的最小値を**図 4** に示す．図中のグラフは，図 2 のグラフと同様に，ノード数に対する仮想通信時間の変化を示す．

実機に近い結果を与える BigSimultor の見積もる仮想通信時間（+）と，通信時間の理論的最小値を比較すると，理論的最小値が非現実的な見積りを与えていることは明らかである．MFS の仮想通信時間（×）は，BigSimulator の仮想通信時間を 100%としたとき，平

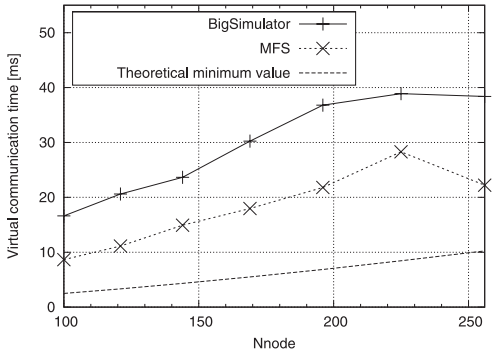


図 4　BigSimulator と MFS の全対全通信における仮想通信時間と全対全通信の理論的最小値との比較
Fig. 4　Comparison of virtual communication time of BigSimulator, MFS, and theoretical minimum value for All-to-all.

⋆1 文献 13) では，旧名称の BigNetSim が使われている．

均で約 40%短かった．また，MFS の結果は，理論的最小値よりもつねに BigSimulator に近かった．以上の結果から，実機に近いという意味での精度を比較すると，MFS の精度は BigSimulator よりも低いといえる．しかし，理論的最小値よりは，BigSimulator や実機に近い見積りを与えていてることが分かった．

256 ノードの通信時間において，BigSimulator と MFS の仮想通信時間はともに短くなっている．これは，2 のべきである 256 ノードの通信に，他の場合に適用された Simple-Spread 法より競合の少ない通信を行う PaireWise 法が適用されたためである．

MFS と BigSimulator の結果に生じた差について考察する．今回行った BigSimulator による測定では，メッセージがネットワーク中を流れている時間のみを比較するため，メッセージサイズが大きな場合と小さな場合におけるシミュレーション結果の差を BigSimulator の見積もる仮想通信時間とした．この測定方法により，ノードで行われる固定長の時間については相殺することができる．一方，送受信バッファへのデータのコピーなど，通信の量に比例した処理が発生すると，これに要する処理時間の一部は相殺できない．この相殺しきれなかった時間を考慮すると，MFS の結果は BigSimulator により近くなると考える．また，MFS と BigSimulator のモデルの違いも，差が生じた原因の 1 つであると考える．

5.3　シミュレーション実行時間

シミュレーション実行時間の測定結果を**表 2** にまとめる．表から，すべてのノード数において，MFS の実行時間は 7 コアで実行した BigSimulator より短いことが分かる．このとき，MFS の実行時間は BigSimulator の約 1/127（$Nnode = 225$）から約 1/337（$Nnode = 256$），平均すると約 1/188 であった．この結果から，MFS のフローに基づくシミュレーションは，BigSimulator のパケットレベルシミュレーションより，実行時間を大幅に短縮することが可能であることが分かった．

BigSimulator の実行時間は，ノード数に対して単調増加であった．一方，MFS の実行時間は $Nnode = 225$ までは単調増加であるが，$Nnode = 256$ では短くなっている．今回の実験では，$Nnode = 256$ の場合，どちらのシミュレータでも，通信アルゴリズムとして Pairewise 法が適用されている．このアルゴリズムによる通信では，通信競合が，他の場合に適用されている Simple-Spread 法と比較して，ネットワーク全体で一様に発生し，かつその変化が少ない．BigSimulator はこのような場合でもパケットの動きを個々に追跡するため，その実行時間はパケット数に応じて増加する．これに対して，MFS は流量に基づくシミュレーションを行っているため，通信競合が一様で，その変化があまりない場合，図 1 の Step 7 で各フローが要求する送信時間 mt_i の最小値として決定される 1 サイクルの時間 Δt が，mt_i の平均に近くなる．これにより，全メッセージの送信がばらつくことなく，整然と行われるため，シミュレーションは短時間で終了する．このように，MFS は，通信競合がネットワーク中で一様に発生する通信アルゴリズムのシミュレーションを短時間で実行できるという性質もある．

表 2　MFS と BigSimulator（7 コア実行）のシミュレーション実行時間の比較
Table 2　MFS vs. BigSimulator (7 CPU core) for simulation run time.

$Nnode$	MFS [s]	BigSimulator (7 CPU core) [s]
100	0.06	11.79
121	0.09	16.25
144	0.14	23.10
169	0.19	31.48
196	0.31	42.43
225	0.44	56.17
256	0.20	67.27

6. 全対全通信アルゴリズムの性能比較

6.1　実 験 方 法

MFS によるシミュレーション結果が，通信アルゴリズムの評価に利用可能であること確認するため，実験を行った．実験は，MFS と BigSimulator それぞれで，通信アルゴリズムを変えたときの仮想通信時間を測定し，その結果を比較するものである．比較には，BigSimulator を再び用いた．実験方法および条件は 5.1 節で述べたものと同様である．通信パターンにも同様に，全対全通信を用いる．

6.2　全対全通信アルゴリズム

全対全通信アルゴリズムとして，A2ASS，A2ASS2D，A2APW を用いる．各通信アルゴリズムでの送信順序のイメージを明確にするため，4×4 の 2 次元 Torus 上の全対全通信において，各アルゴリズムを適用したとき，1 番ノードがどのような順番でメッセージを送信するのかを具体的に**図 5** に示す．この図を用いて，それぞれのアルゴリズムの概略を説明する．

A2ASS は送信ノード src から p 番目に送信するメッセージの受信ノード番号 $dest$ を

$$dest = (src + p)\%n^2 \tag{1}$$

として決定する通信アルゴリズムである．ただし，$0 \leq src$，$dest < n^2$，$1 \leq p < n^2$ である．また，% は剰余演算を表す．この方法は，Simple-Spread 法として知られている．図 5

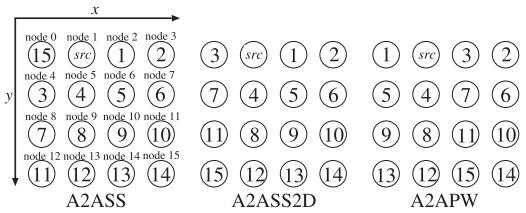


図 5　A2ASS，A2ASS2D，A2APW 各アルゴリズムの送信順序（$n = 4$，$src = 1$）
Fig. 5　Sending order for A2ASS, A2ASS2D and A2APW All-to-all communication algorithms in case of $n = 4, src = 1$.

中の左に，$n = 4$，$src = 1$ における具体例を示す．円の中の番号は送信順番 p を意味する．この通信アルゴリズムでは，1 回目の送信で，x 方向に隣，2 回目の送信ではさらに隣という順番で送信を行う．x 方向の最後まで達したら，次は 1 つ下の段に関して同様の順番で送信を行う．全ノードがこのような送信を同時に行うので，通信の重なりは多い．

A2ASS2D は，受信ノード番号 $dest$ を

$$dest = \{(src\%n + p\%n)\%n\} + \{(\lfloor src/n \rfloor + \lfloor p/n \rfloor)\%n\} \times n \quad (2)$$

として求める通信アルゴリズムである．A2ASS2D は，Simple-Spread 法を 2 次元に拡張し，次元ごとに Simple-Spread 法を適用したものである．これは，Simple-Spread 法を Torus や Mesh ネットワークに適応させた方法であるといえる．図 5 中の中央に通信順序の例を示す．

A2APW は受信ノード番号 $dest$ を

$$dest = (src \oplus p) \quad (3)$$

として求めている．ただし，$\oplus$ は排他論理和演算を表す．この方法は，Pairwise 法として知られている．図 5 の右に通信順序の例を示す．この通信アルゴリズムは，ノードペアを作り，ペアどうしで互いに通信をすることで，同一方向への通信を減らし，通信競合を緩和している．ただし，この方法は，ノード数が 2 のべきである場合にしか用いることができない．

6.3 実験結果

図 6 に MFS と BigSimulator で測定した各全対全通信アルゴリズムにおける仮想通信時間を示す．図から，全体全通信の仮想通信時間は，A2ASS，A2ASS2D，A2APW の順に短くなることがわかる．この結果は，各アルゴリズムの定性的な性質と一致する．A2ASS は，今回用いた通信アルゴリズムの中で最も通信経路競合が起きやすいアルゴリズムである．A2ASS では，全ノードが同時に同じ向きに通信を行うため，ノードを接続している双

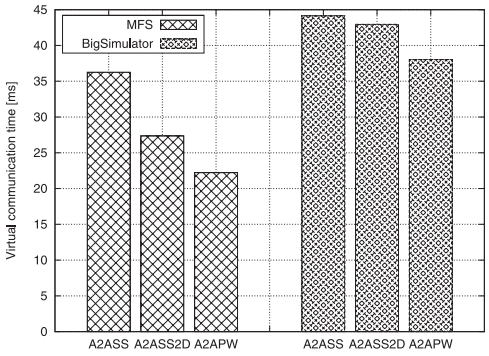


図 6　全対全通信アルゴリズムの性能差比較
Fig. 6　Comparison of performences of communication algorithms for All-to-all.

方向リンクの，上り方向を多くのノードが同時に使う場面や，下り方向を多くのノードが同時に使う場面が多く発生してしまう．A2ASS2D は，A2ASS より効率的にリンクを利用することができるアルゴリズムである．A2ASS2D は，送信ノードを中心として，x 方向の左右にメッセージを送信した後，y 方向に通信を行う．これにより，双方向リンクの上りと下りを効率的に利用することができる．A2APW は，ノードペアを作り，ペアどうしで通信を行うため，双方向リンクをさらに効率良く使うことができる．

図から MFS と BigSimulator の結果を比較すると，各通信アルゴリズムによる仮想通信時間の大小関係が一致していることが分かる．これは，MFS が採用するフローに基づくシミュレーション方式が BigSimulator が採用するパケットレベルでのシミュレーションと同様に，通信アルゴリズムの性能を評価できることを示している．

7. ま と め

本論文では，通信アルゴリズムの評価を高速に行うことを目的として，従来のパケットレベルシミュレータより高速にシミュレーションを実行することが可能なシミュレータ MFS（Message Flow Simulator）の開発を行った結果について述べた．MFS は，並列計算における通信をパケット単位でシミュレーションせず，フローとして扱うことで高速なシミュレーションを実現している．

MFS の見積もる仮想通信時間とシミュレーション実行時間から，MFS の特性を示す数値実験を行った．実験では，フルバイセクションバンド幅を持つ Fattree 上の全対全通信パ

タンを取り上げ，MFS の特性を評価した．MFS において，ループ処理によってシミュレーションの精度を高める Mode=N と，これを省略し高速化を図る Mode=F のそれぞれについて，シミュレーションによって見積もられた仮想通信時間とシミュレーションの実行時間を示した．

既存のパケットレベルシミュレータと，シミュレーション結果および実行時間を比較する実験を行った．2 次元 Torus ネットワーク上の全対全通信において，実機に近いシミュレーションを行うパケットレベルシミュレータ BigSimulator と MFS を比較した．MFS の仮想通信時間は，ノード数 100 から 256 において，BigSimulator の仮想通信時間に対して平均 40%短かった．この実験から，MFS のシミュレーション精度は BigSimulator よりも低いが，現実的でない見積りを与える通信の理論的最小値よりは BigSimulator や実機に近い結果を与えることを示した．今回行った BigSimulator による測定では，ノードで行われる通信の量に応じた処理に要する時間の一部は相殺できなかった．これを考慮すると，MFS の結果は BigSimulator に，より近くなると考えられる．MFS のシミュレーション実行時間は BigSimulator の約 1/127（$Nnode = 225$）から約 1/337（$Nnode = 256$），平均で約 1/188 であった．16×16 の 2 次元 Torus 上で，異なる 3 種類の全対全通信アルゴリズム A2ASS，A2ASS2D，A2APW について，MFS と BigSimulator の両方でその性能差を同様に評価できることを示した．

今後は，通信アルゴリズムの評価における MFS の有効性を他のシミュレータとさらに比較する必要がある．今回の実験では，BigSimulator の測定において，ノード側の処理の影響を完全に排除できなかった．今後は，これを排除し，メッセージがネットワークを流れている時間だけに着目した通信時間の比較を行う必要がある．

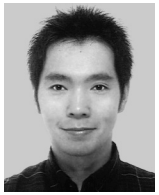
謝辞　本研究を進めるにあたりご協力いただいた富士通株式会社次世代テクニカルコンピューティング開発本部の追永勇次氏，清水俊幸氏に深謝します．本研究は，九州大学情報基盤研究センターの研究用計算機システムを利用して行われました．

参 考 文 献

1) http://www.jamstec.go.jp/esc/index.en.html
2) Hoisie, A., et al.: A Performance Comparison Through Benchmarking and Modeling of Three Leading Supercomputers: Blue Gene/L, Red Storm and Purple, *SC '06: Proc. 2006 ACM/IEEE conference on Supercomputing*, p.3 (2006).
3) Alam, S.R., et al.: Cray XT4: an early evaluation for petascale scientific simulation, *SC '07: Proc. 2007 ACM/IEEE conference on Supercomputing*, New York, NY, USA, ACM, pp.1–12 (2007).
4) Barker, K.J., et al.: Entering the petaflop era: the architecture and performance of Roadrunner, *SC '08: Proc. 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, pp.1–11, IEEE Press (2008).
5) Ang, B.S., et al.: Micro-architectures of high performance, multi-user system areanetwork interface cards, *Proc. IPDPS 2000*, pp.13–20 (2000).
6) Rexford, J., et al.: PP-MESS-SIM: A Simulator for Evaluating Multicomputer Interconnection Networks, *Proc. 28th Annual Simulation Symposium*, p.84 (1995).
7) Berkbigler, K., et al.: A la carte: A simulation framework for extreme-scale hardware architectures, *Proc. IASTED International Conference on Modelling and Simulation*, CA, USA, pp.38–43 (2003).
8) 若林正樹，天野英晴：並列計算機シミュレータの構築支援環境，電子情報通信学会論文誌 D-I，Vol.J84-D-I, pp.247–256 (2001).
9) Boku, T., et al.: INSPIRE: A generalpurpose network simulator generating system for massively parallel processors, *Proc. PER-MEAN95*, pp.24–33 (1999).
10) Choudhury, N., et al.: Scaling an optimistic parallel simulation of large-scale interconnection networks, *Proc. WSC '05, Winter Simulation Conference*, pp.591–600 (2005).
11) http://charm.cs.uiuc.edu/research/bignetsim/
12) Wilmarth, T.L., et al.: Performance Prediction Using Simulation of Large-Scale Interconnection Networks in POSE, *Proc. PADS '05*, Washington, DC, USA, pp.109–118, IEEE Computer Society (2005).
13) Wilmarth, T.L., et al.: Performance Prediction Using Simulation of Large-Scale Interconnection Networks in POSE, *PADS '05: Proc. 19th Workshop on Principles of Advanced and Distributed Simulation*, Washington, DC, USA, pp.109–118, IEEE Computer Society (2005).
14) 田中良夫ほか：Collective 通信を用いたデータ並列プログラムの性能予測，情報処理学会研究報告 ハイパフォーマンスコンピューティング，Vol.65, pp.69–74 (1997).
15) 久保田和人ほか：大規模データ並列プログラムの性能予測手法と NPB 2.3 の性能評価，情報処理学会論文誌，Vol.40, pp.2293–2303 (1999).
16) Susukita, R., et al.: Performance Prediction of Large-scale Parallel System and Application using Macro-level Simulation, *SC08: Proc. International Conference for High Performance Computing, Networking, Storage and Analysis* (2008).
17) 神戸友樹ほか：メッセージ通信プログラムの改善可能性を評価するための性能予測ツール，情報処理学会論文誌：コンピューティングシステム，Vol.44, pp.101–110 (2003).
18) 三島　健ほか：超並列計算機用多段結合網における転送性能の解析，情報処理学会論文誌，Vol.40, pp.1985–1995 (1999).
19) 天方貴久ほか：A Simulator for Message-Passing Based Parallel and Distributed

Programs to Evaluate the Influence of Message Transfer Latency, 情報処理学会研究報告 MPS, Vol.45, pp.25–28 (2003).
20) Nicol, D.M.: Fluid simulation: discrete event fluid modeling of TCP, *Proc. WSC '01*, Washington, DC, USA, pp.1291–1299, IEEE Computer Society (2001).
21) Nicol, D.M. and Yan, G.: Simulation of Network Traffic at Coarse Timescales, *Proc. PADS '05*, Washington, DC, USA, pp.141–150, IEEE Computer Society (2005).
22) 西岡孟朗ほか：異種のフローが混在するネットワークに対応したフローレベルシミュレータの設計と実装，電子情報通信学会技術研究報告，Vol.108, No.136, pp.65–70 (2008).
23) http://www-unix.mcs.anl.gov/mpi/

(平成 21 年 10 月 2 日受付)
(平成 22 年 2 月 16 日採録)



矢崎　俊志

2007 年電気通信大学大学院電気通信学研究科情報工学専攻博士後期課程修了．2009 年東京工科大学助教．2010 年電気通信大学情報基盤センター助教．並列計算機，生活支援システム，算術論理演算回路に関する研究に従事．博士（工学）．信学会，パルテノン研究会各会員．

石畑　宏明（正会員）

1980 年早稲田大学理工学部卒業．同年（株）富士通研究所入社．2007 年東京工科大学教授．並列コンピュータアーキテクチャの研究に従事．1992 年元岡賞受賞，博士（工学）．信学会，IEEE 各会員．